Prolog Programming Assignment #2:

State Space Problem Solving

Thomas Moskal

CSC 344 – Professor Graci

Due date: 3/22/22

Abstract

In this assignment build up a program that will eventually create a valid solution to the Towers of Hanoi problem. This is an interesting program that uses Prolog's backend problem solving in very cool ways. We slowly place pieces of this code together until we have one that can solve level the problem with 5 disks. We do this by creating moves, valid states, goal and start states. The program does not resolve the problem with the perfect solution… so far.

Task 3: One Move Predicate and a Unit Test

State Space Operator Implementation Code:

```
m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-
Tower1Before = [H|T],
Tower1After = T,
Tower2Before = L,
Tower2After = [H|L].
```

Unit Test Code:

```
test__m12 :-
write('Testing: move_m12\n'),TowersBefore = [[t,s,m,l,h],[],[]],
trace('','TowersBefore',TowersBefore),
m12(TowersBefore,TowersAfter),
trace('','TowersAfter',TowersAfter).
```

Unit Test Demo:

```
3 ?- test__m12
.
Testing: move_m12
TowersBefore = [[t,s,m,l,h],[],[]]
TowersAfter = [[s,m,l,h],[t],[]]
true.
```

Task 4: The Remaining Five Move Predicates and a Unit Tests

State Space Operators Code:

```
% ------------------------------------------------------------------------
% --- State Space operators

m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-
Tower1Before = [H|T],
Tower1After = T,
Tower2Before = L,
Tower2After = [H|L].

m13([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-
Tower1Before = [H|T],
Tower1After = T,
Tower3Before = L,
Tower3After = [H|L].

m21([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-
Tower2Before = [H|T],
Tower2After = T,
Tower1Before = L,
Tower1After = [H|L].

m23([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-
Tower2Before = [H|T],
Tower2After = T,
Tower3Before = L,
Tower3After = [H|L].

m31([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-
Tower3Before = [H|T],
Tower3After = T,
Tower1Before = L,
Tower1After = [H|L].

m32([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-
Tower3Before = [H|T],
Tower3After = T,
Tower2Before = L,
Tower2After = [H|L].
```

Unit Test Programs Code:

```
% ------------------------------------------------------------------------
% --- State Space operator tests

test__m12 :-
write('Testing: move_m12\n'),TowersBefore = [[t,s,m,l,h],[],[]],
trace('','TowersBefore',TowersBefore),
m12(TowersBefore,TowersAfter),
trace('','TowersAfter',TowersAfter).

test__m13 :-
write('Testing: move_m13\n'),TowersBefore = [[t,s,m,l,h],[],[]],
trace('','TowersBefore',TowersBefore),
m13(TowersBefore,TowersAfter),
trace('','TowersAfter',TowersAfter).

test__m21 :-
write('Testing: move_m21\n'),TowersBefore = [[],[t,s,m,l,h],[]],
trace('','TowersBefore',TowersBefore),
m21(TowersBefore,TowersAfter),
trace('','TowersAfter',TowersAfter).

test__m23 :-
write('Testing: move_m23\n'),TowersBefore = [[],[t,s,m,l,h],[]],
trace('','TowersBefore',TowersBefore),
m23(TowersBefore,TowersAfter),
trace('','TowersAfter',TowersAfter).

test__m31 :-
write('Testing: move_m31\n'),TowersBefore = [[],[],[t,s,m,l,h]],
trace('','TowersBefore',TowersBefore),
m31(TowersBefore,TowersAfter),
trace('','TowersAfter',TowersAfter).

test__m32 :-
write('Testing: move_m32\n'),TowersBefore = [[],[],[t,s,m,l,h]],
trace('','TowersBefore',TowersBefore),
m32(TowersBefore,TowersAfter),
trace('','TowersAfter',TowersAfter).
```

Unit Test Programs Demo:

```
5 ?- test__m12.
Testing: move_m12
TowersBefore = [[t,s,m,l,h],[],[]]
TowersAfter = [[s,m,l,h],[t],[]]
true.

6 ?- test__m13.
Testing: move_m13
TowersBefore = [[t,s,m,l,h],[],[]]
TowersAfter = [[s,m,l,h],[],[t]]
true.

7 ?- test__m21.
Testing: move_m21
TowersBefore = [[],[t,s,m,l,h],[]]
TowersAfter = [[t],[s,m,l,h],[]]
true.

8 ?- test__m23.
Testing: move_m23
TowersBefore = [[],[t,s,m,l,h],[]]
TowersAfter = [[],[s,m,l,h],[t]]
true.

9 ?- test__m31.
Testing: move_m31
TowersBefore = [[],[],[t,s,m,l,h]]
TowersAfter = [[t],[],[s,m,l,h]]
true.

10 ?- test__m32.
Testing: move_m32
TowersBefore = [[],[],[t,s,m,l,h]]
TowersAfter = [[],[t],[s,m,l,h]]
true.
```

Task 5: Valid State Predicate and Unit Test

valid_state Predicate Code:

```
% ------------------------------------------------------------------------
% --- valid_state(S) :: S is a valid state

valid_state([A | [B | [C]]]) :- state(A), state(B), state(C).

state([]).
state([t]).
state([t,s]).
state([t,s,m]).
state([t,s,m,l]).
state([t,s,m,h]).
state([t,s,m,l,h]).
state([t,s,l]).
state([t,s,h]).
state([t,s,l,h]).
state([t,m]).
state([t,m,l]).
state([t,m,l,h]).
state([t,l]).
state([t,l,h]).
state([t,h]).
state([s]).
state([s,m]).
state([s,m,l]).
state([s,m,l,h]).
state([s,l]).
state([s,l,h]).
state([s,h]).
state([m]).
state([m,l]).
state([m,l,h]).
state([m,h]).
state([l]).
state([l,h]).
state([h]).
```

Unit Test Program Code:

```
% ----------------------------------------------------------------------
% --- valid_state tests

test__valid_state :-
    write('Testing: vaid_state\n'),
    test__vs([[l,t,s,m,h],[],[]]),
    test__vs([[t,s,m,l,h],[],[]]),
    test__vs([[],[h,t,s,m],[l]]),
    test__vs([[],[t,s,m,h],[l]]),
    test__vs([[],[h],[l,m,s,t]]),
    test__vs([[],[h],[t,s,m,l]]).

test__vs(S) :-
    valid_state(S),
    write(S), write('i is valid.'), nl.
test__vs(S) :-
    write(S), write(' is invalid.'), nl.
```

Unit Test Program Demo:

```
11 ?- test__valid_state.
Testing: vaid_state
[[l,t,s,m,h],[],[]] is invalid.
[[t,s,m,l,h],[],[]]i is valid.
[[],[h,t,s,m],[l]] is invalid.
[[],[t,s,m,h],[l]]i is valid.
[[],[h],[l,m,s,t]] is invalid.
[[],[h],[t,s,m,l]]i is valid.
true
```

Task 6: Defining the write sequence predicate

write_sequence Predicate Code:

```
% ----------------------------------------------------------------------
% --- write_sequence


write_sequence([]).

write_sequence([H|T]) :-
    sequence(H,Explanation),
    write(Explanation),nl,
    write_sequence(T).

sequence(m12, Explanation) :-
    Explanation = 'Transfer a disk from tower 1 to 2.'.
sequence(m13, Explanation) :-
    Explanation = 'Transfer a disk from tower 1 to 3.'.
sequence(m21, Explanation) :-
    Explanation = 'Transfer a disk from tower 2 to 1.'.
sequence(m23, Explanation) :-
    Explanation = 'Transfer a disk from tower 2 to 3.'.
sequence(m31, Explanation) :-
    Explanation = 'Transfer a disk from tower 3 to 1.'.
sequence(m32, Explanation) :-
    Explanation = 'Transfer a disk from tower 3 to 2.'.
```

Unit Tester Code:

```
test__write_sequence :-
    write('First test of write_sequence ...'), nl,
    write_sequence([m31,m12,m13,m21]),
    write('Second test of write_sequence ...'), nl,
    write_sequence([m13,m12,m32,m13,m21,m23,m13]).
```

Unit Tester Demo:

```
12 ?- test__write_sequence.
First test of write_sequence ...
Transfer a disk from tower 3 to 1.
Transfer a disk from tower 1 to 2.
Transfer a disk from tower 1 to 3.
Transfer a disk from tower 2 to 1.
Second test of write_sequence ...
Transfer a disk from tower 1 to 3.
Transfer a disk from tower 1 to 2.
Transfer a disk from tower 3 to 2.
Transfer a disk from tower 1 to 3.
Transfer a disk from tower 2 to 1.
Transfer a disk from tower 2 to 3.
Transfer a disk from tower 1 to 3.
true.
```

Task 7: Run the Program to Solve the 3 Disk Problem

Intermediate output display demo:

```
14 ?- solve.
PathSoFar = [[[s,m,l],[],[]]]
Move = m12
NextState = [[m,l],[s],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]]]
Move = m12
NextState = [[l],[m,s],[]]
Move = m13
NextState = [[l],[s],[m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]]]
Move = m12
NextState = [[],[l,s],[m]]
Move = m13
NextState = [[],[s],[l,m]]
Move = m21
NextState = [[s,l],[],[m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]]]
Move = m12
NextState = [[l],[s],[m]]
Move = m13
NextState = [[l],[],[s,m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]]]
Move = m12
NextState = [[],[l],[s,m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]]]
Move = m21
NextState = [[l],[],[s,m]]
Move = m23
NextState = [[],[],[l,s,m]]
Move = m31
NextState = [[s],[l],[m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]]]
Move = m12
NextState = [[],[s,l],[m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]]]
```

```
Move = m21
NextState = [[s],[l],[m]]
Move = m23
NextState = [[],[l],[s,m]]
Move = m31
NextState = [[m],[s,l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]]]
Move = m12
NextState = [[],[m,s,l],[]]
Move = m13
NextState = [[],[s,l],[m]]
Move = m21
NextState = [[s,m],[l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]]]
Move = m12
NextState = [[m],[s,l],[]]
Move = m13
NextState = [[m],[l],[s]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[m],[l
,[s]]]
Move = m12
NextState = [[],[m,l],[s]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[m],[l
,[s]],[[],[m,l],[s]]]
Move = m21
NextState = [[m],[l],[s]]
Move = m23
NextState = [[],[l],[m,s]]
Move = m31
NextState = [[s],[m,l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[m],[l
,[s]],[[],[m,l],[s]],[[s],[m,l],[]]]
Move = m12
NextState = [[],[s,m,l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[m],[l
,[s]],[[],[m,l],[s]],[[s],[m,l],[]],[[],[s,m,l],[]]]
```

```
Move = m23
NextState = [[],[m,l],[s]]
Move = m13
NextState = [[],[m,l],[s]]
Move = m21
NextState = [[m,s],[l],[]]
Move = m23
NextState = [[s],[l],[m]]
Move = m32
NextState = [[],[s,m,l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[m],[l
,[s]],[[],[m,l],[s]],[[],[s,m,l],[]]]
Move = m21
NextState = [[s],[m,l],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[m],[l
,[s]],[[],[m,l],[s]],[[],[s,m,l],[]],[[s],[m,l],[]]]
Move = m12
NextState = [[],[s,m,l],[]]
Move = m13
NextState = [[],[m,l],[s]]
Move = m21
NextState = [[m,s],[l],[]]
Move = m23
NextState = [[s],[l],[m]]
Move = m23
NextState = [[],[m,l],[s]]
Move = m13
NextState = [[],[l],[m,s]]
Move = m21
NextState = [[1,m],[],[s]]
Move = m23
NextState = [[m],[],[l,s]]
Move = m31
NextState = [[s,m],[l],[]]
Move = m32
NextState = [[m],[s,l],[]]
Move = m21
NextState = [[l,s,m],[],[]]
Move = m23
```

```
NextState = [[s,m],[],[l]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[s,m],[],[l]]]
Move = m12
NextState = [[m],[s],[l]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[s,m],[],[l]],[[m],[s],[l]]]
Move = m12
NextState = [[],[m,s],[l]]
Move = m13
NextState = [[],[s],[m,l]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[s,m],[],[l]],[[m],[s],[l]],[[],[s],[m,l]]]
Move = m21
NextState = [[s],[],[m,l]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[s,m],[],[l]],[[m],[s],[l]],[[],[s],[m,l]],[[s],[],[m,l]]]
Move = m12
NextState = [[],[s],[m,l]]
Move = m13
NextState = [[],[],[s,m,l]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,l],[]],[[s,m],[l],[]],[[s,m],[],[l]],[[m],[s],[l]],[[],[s],[m,l]],[[s],[],[m,l]],[[],[],[s,m,l]]]
SolutionSoFar = [m12,m13,m21,m13,m12,m31,m12,m31,m21,m23,m12,m13,m21,m13]
```

English only solution display demo:

```
Solution ...

Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 2 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
```

Task 7 Q&A:

1. What was the length of your program's solution to the three disk problem?

    - My program's solution was 14 moves

2. What is the length of the shortest solution to the three disk problem?

    -The shortest solution to the three disk problem is 7.

3. How do you account for the discrepancy?

    -The program is constrained by not being able to think ahead. It makes moves mindlessly, constrained by the rules of its code, until it finds a valid solution. It is unaware of the most efficient path to the goal state, but it will always find a valid one.

Task 8: Run the Program to Solve the 4 Disk Problem

4 Disk English Solution Demo:

```
Solution ...

Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
```

Task 8 Q&A:

1. What was the length of your program's solution to the four disk problem?

    - My program's solution length was 40 moves.

2. What is the length of the shortest solution to the four disk problem?

    -The shortest 4 disk problem solution is 15 moves.

Task 9: Review Your Code and Archive it

Towers of Hanoi Code

```
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% --- File: towers_of_hanoi.pro
% --- Line: Program to solve the Towers of Hanoi problem
% ------------------------------------------------------------------------
:- consult('inspectors.pro').
% ------------------------------------------------------------------------
% --- make_move(S,T,SSO) :: Make a move from state S to state T by SSO

make_move(TowersBeforeMove,TowersAfterMove,m12) :-
m12(TowersBeforeMove,TowersAfterMove).

make_move(TowersBeforeMove,TowersAfterMove,m13) :-
m13(TowersBeforeMove,TowersAfterMove).

make_move(TowersBeforeMove,TowersAfterMove,m21) :-
m21(TowersBeforeMove,TowersAfterMove).

make_move(TowersBeforeMove,TowersAfterMove,m23) :-
m23(TowersBeforeMove,TowersAfterMove).

make_move(TowersBeforeMove,TowersAfterMove,m31) :-
m31(TowersBeforeMove,TowersAfterMove).

make_move(TowersBeforeMove,TowersAfterMove,m32) :-
m32(TowersBeforeMove,TowersAfterMove).

```

```
27
28  % ---------------------------------------------------------------------------
29  % --- valid_state(S) :: S is a valid state
30
31  valid_state([A | [B | [C]]]) :- state(A), state(B), state(C).
32
33  state([]).
34  state([t]).
35  state([t,s]).
36  state([t,s,m]).
37  state([t,s,m,l]).
38  state([t,s,m,h]).
39  state([t,s,m,l,h]).
40  state([t,s,l]).
41  state([t,s,h]).
42  state([t,s,l,h]).
43  state([t,m]).
44  state([t,m,l]).
45  state([t,m,l,h]).
46  state([t,l]).
47  state([t,l,h]).
48  state([t,h]).
49  state([s]).
50  state([s,m]).
51  state([s,m,l]).
52  state([s,m,l,h]).
53  state([s,l]).
54  state([s,l,h]).
55  state([s,h]).
56  state([m]).
57  state([m,l]).
58  state([m,l,h]).
59  state([m,h]).
60  state([l]).
61  state([l,h]).
62  state([h]).
63
```

```
64  % ------------------------------------------------------------------------
65  % --- valid_state tests
66
67  test__valid_state :-
68      write('Testing: vaid_state\n'),
69      test__vs([[l,t,s,m,h],[],[]]),
70      test__vs([[t,s,m,l,h],[],[]]),
71      test__vs([[],[h,t,s,m],[l]]),
72      test__vs([[],[t,s,m,h],[l]]),
73      test__vs([[],[h],[l,m,s,t]]),
74      test__vs([[],[h],[t,s,m,l]]).
75
76  test__vs(S) :-
77      valid_state(S),
78      write(S), write('i is valid.'), nl.
79  test__vs(S) :-
80      write(S), write(' is invalid.'), nl.
81
82
83  % ------------------------------------------------------------------------
84  % --- solve(Start,Solution) :: succeeds if Solution represents a path
85  % --- from the start state to the goal state.
86
87  solve :-
88  extend_path([[[t,s,m,l],[],[]]],[],Solution),
89  write_solution(Solution).
90
91  extend_path(PathSoFar,SolutionSoFar,Solution) :-
92  PathSoFar = [[[],[],[t,s,m,l]]|_],
93  showr('PathSoFar',PathSoFar),
94  showr('SolutionSoFar',SolutionSoFar),
95  Solution = SolutionSoFar.
96
97  extend_path(PathSoFar,SolutionSoFar,Solution) :-
98  PathSoFar = [CurrentState|_],
99  showr('PathSoFar',PathSoFar),
100 make_move(CurrentState,NextState,Move),
101 show('Move',Move),
102 show('NextState',NextState),
103 not(member(NextState,PathSoFar)),
104 valid_state(NextState),
105 Path = [NextState|PathSoFar],
106 Soln = [Move|SolutionSoFar],extend_path(Path,Soln,Solution).
107
```

```
% ---------------------------------------------------------------------
% --- write_sequence


write_sequence([]).

write_sequence([H|T]) :-
    sequence(H,Explanation),
    write(Explanation),nl,
    write_sequence(T).

sequence(m12, Explanation) :-
    Explanation = 'Transfer a disk from tower 1 to tower 2.'.
sequence(m13, Explanation) :-
    Explanation = 'Transfer a disk from tower 1 to tower 3.'.
sequence(m21, Explanation) :-
    Explanation = 'Transfer a disk from tower 2 to tower 1.'.
sequence(m23, Explanation) :-
    Explanation = 'Transfer a disk from tower 2 to tower 3.'.
sequence(m31, Explanation) :-
    Explanation = 'Transfer a disk from tower 3 to tower 1.'.
sequence(m32, Explanation) :-
    Explanation = 'Transfer a disk from tower 3 to tower 2.'.


test__write_sequence :-
    write('First test of write_sequence ...'), nl,
    write_sequence([m31,m12,m13,m21]),
    write('Second test of write_sequence ...'), nl,
    write_sequence([m13,m12,m32,m13,m21,m23,m13]).


% ---------------------------------------------------------------------
% --- write_sequence_reversed(S) :: Write the sequence, given by S,
% --- expanding the tokens into meaningful strings.

write_solution(S) :-
nl, write('Solution ...'), nl, nl,
reverse(S,R),
write_sequence(R),nl.

```

```
149  % ------------------------------------------------------------------------
150  % --- State Space operators
151
152  m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-
153  Tower1Before = [H|T],
154  Tower1After = T,
155  Tower2Before = L,
156  Tower2After = [H|L].
157
158  m13([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-
159  Tower1Before = [H|T],
160  Tower1After = T,
161  Tower3Before = L,
162  Tower3After = [H|L].
163
164  m21([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]) :-
165  Tower2Before = [H|T],
166  Tower2After = T,
167  Tower1Before = L,
168  Tower1After = [H|L].
169
170  m23([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-
171  Tower2Before = [H|T],
172  Tower2After = T,
173  Tower3Before = L,
174  Tower3After = [H|L].
175
176  m31([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]) :-
177  Tower3Before = [H|T],
178  Tower3After = T,
179  Tower1Before = L,
180  Tower1After = [H|L].
181
182  m32([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]) :-
183  Tower3Before = [H|T],
184  Tower3After = T,
185  Tower2Before = L,
186  Tower2After = [H|L].
187
```

```
188   % ----------------------------------------------------------------------
189   % --- State Space operator tests
190
191   test__m12 :-
192   write('Testing: move_m12\n'),TowersBefore = [[t,s,m,l,h],[],[]],
193   trace('','TowersBefore',TowersBefore),
194   m12(TowersBefore,TowersAfter),
195   trace('','TowersAfter',TowersAfter).
196
197   test__m13 :-
198   write('Testing: move_m13\n'),TowersBefore = [[t,s,m,l,h],[],[]],
199   trace('','TowersBefore',TowersBefore),
200   m13(TowersBefore,TowersAfter),
201   trace('','TowersAfter',TowersAfter).
202
203   test__m21 :-
204   write('Testing: move_m21\n'),TowersBefore = [[],[t,s,m,l,h],[]],
205   trace('','TowersBefore',TowersBefore),
206   m21(TowersBefore,TowersAfter),
207   trace('','TowersAfter',TowersAfter).
208
209   test__m23 :-
210   write('Testing: move_m23\n'),TowersBefore = [[],[t,s,m,l,h],[]],
211   trace('','TowersBefore',TowersBefore),
212   m23(TowersBefore,TowersAfter),
213   trace('','TowersAfter',TowersAfter).
214
215   test__m31 :-
216   write('Testing: move_m31\n'),TowersBefore = [[],[],[t,s,m,l,h]],
217   trace('','TowersBefore',TowersBefore),
218   m31(TowersBefore,TowersAfter),
219   trace('','TowersAfter',TowersAfter).
220
221   test__m32 :-
222   write('Testing: move_m32\n'),TowersBefore = [[],[],[t,s,m,l,h]],
223   trace('','TowersBefore',TowersBefore),
224   m32(TowersBefore,TowersAfter),
225   trace('','TowersAfter',TowersAfter).
```

Inspectors.pro Code

```
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% --- File: inspectors.pro
% --- Line: Utilities for inspecting memory during program execution
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% --- These two can be used to print the value of a variable, labelled
% --- in two ways, and pause for the programmer to check out the
% --- situation. The firstone is generally useful. The second one is
% --- applicable only whenthe value of the veriable is a list, and it
% --- will print the valuein reverse order which is sometimes just
% --- what is desired. The first label generally pertains to a location
% --- in the program. The second label is just the name of the variable
% --- to which the value is bound.

check(Label,Name,Value) :-
write(Label),
write(Name),write(' = '),
write(Value),nl,
read(_).

checkr(Label,Name,Value) :-
write(Label),
write(Name),write(' = '),
reverse(Value,RValue),
write(RValue),nl,
read(_).

% ------------------------------------------------------------------------
% --- These two are like the previously described checking predicates,
% --- except that they do not do the pause.

trace(Label,Name,Value) :-
write(Label),
write(Name),write(' = '),
write(Value),nl.
tracer(Label,Name,Value) :-
write(Label),
write(Name),write(' = '),
reverse(Value,RValue),
write(RValue),nl.

% ------------------------------------------------------------------------
% --- Like trace, but without the extra labelling functionality.

show(Name,Value) :-
write(Name),write(' = '),
write(Value),nl.
showr(Name,Value) :-
write(Name),write(' = '),
reverse(Value,RValue),
write(RValue),nl.
```